Universitat de Lleida

TREBALL FINAL DE MÀSTER

Estudiant: **Javier Bonet Lambea**

Titulació: Màster en Enginyeria Informàtica

Títol de Treball Final de Màster: Building a Chatbot for Health Care Patients Quitting Smoking

Director/a: Jordi Vilaplana

Presentació

Mes: Juny

Any: 2019

Building a Chatbot for Health Care Patients Quitting Smoking

Javier Bonet

Abstract—The aim of this study is to build and assess the viability of a chatbot for patients who are quitting smoking. Chatbots are everywhere and the tools for building one are widely available now. We are going to explore how to build a bot that will act as a virtual therapist, helping patients who are quitting smoking. By providing the patients with a tool to get the help they need without going to the doctor's office we intend to make patients feel more confident. The tool is integrated with several conversational interfaces so patients can get access to the chatbot from the platform of their choice. The study results show that even though building a working chatbot is now really easy, building a virtual therapist which understands the needs of a patient, however, has many challenges to overcome yet.

Index Terms—Chatbots, Smoking, Health care, Natural Language

1 Introduction

For a long time computer scientists have tried to create programs that can talk to and understand humans. First attempts were made over fifty years ago using really simple rule systems that tried to extract context from the user input, creating the illusion of understanding.

Those early systems have evolved to Natural Language Understanding (NLU). NLU is a field of computer science which focuses, with computational techniques, in understanding human language. Thanks to cloud computing, there are now several NLU platforms that work as a service. Some of the most known platforms are LUIS (Microsoft), Watson (IBM) and Dialogflow (Google) [1]. By using these platforms, we can focus on building the interaction and leaving the language processing and knowledge extraction to them.

Thanks to virtual assistants such as Google Assistant or Amazon Alexa and messaging platforms like Telegram or Facebook Messenger and their integrations with NLU platforms, chatbots are being used by companies for new kind of interactions, replacing their traditional websites for actions such as booking reservations, purchasing goods or customer service [2].

Another area where chatbots are being used is in healthcare. Chatbots in healthcare bring a new way for the patients to have assistance. While some of the existing chatbots aim to replace healthcare professionals [3] our intend is to fill the gap between the patients and the doctors in the times that the patients are alone, allowing doctors to follow up on their patients remotely.

In this study we propose an approach to build a functional voice and text chatbot that works from multiple conversational interfaces. We focus in building a chatbot that supports patients who are quitting smoking and has as interface Google Assistant and Telegram. Being both interchangeable, we do not enforce patients to buy an specific device in order to interact with our chatbot.

2 Related Work

Recent publications have discussed different approaches on how to create a chatbot and its applications. One of the applications is to replace a website [4] with a voice enabled chatbot. The paper describes how the author built a system that could perform the same tasks as the website it was aiming to replace using for the implementation the DialogFlow platform. In the field of healthcare we find how others choose completely different platforms to build their chatbots, like the Microsoft Bot Framework which uses LUIS in background [3]. But as it is shown, it does not perform really well in an open environment, where the users can provide any kind of input. We also have to take into account the results found in 2015 by J. Hill et al. [5], where they performed a thorough comparison on

how users behave when they know they are talking to a bot and not to a human. They found out that when users realize they are not talking to a human, they shorten their responses and have less patience. Knowing this is very important for us in order to provide better responses and adapting the inputs we would expect to shortened versions of them.

3 METHODOLOGY

In this section we present the architecture and implementation of the chatbot. We have chosen the Dialogflow platform as it is the one that has better fallback matching and provides a comprehensive fulfillment library to create rich responses via webhooks, meaning we only have to listen to POST requests and handling them, instead of using a full SDK (Software Development Toolkit). It also has great integration with several Voice Assistants and Chat applications, so we can make our chatbot multiplatform easily.

Dialogflow is a cloud Natural Language Processing platform that allows developers to build rich conversational interfaces in a very simple way. It has integrations with the most common chat platforms and voice assistants such as Assistant, Telegram, Skype or Alexa. Dialogflow offers a lot of functionality for free that in other platforms has a cost of hundreds or even thousands of euros per month. Among its functionalities, we find advanced speech recognition and the ability to use our chatbot by phone.

Webhooks are simple HTTP POST requests that expect a response in an specific format so it can use the response. In our case, we will have an endpoint listening for this requests, and the Dialogflow Library will take care of formatting the responses and sending them back to the agent on Dialogflow.

3.1 Design

3.1.1 Architecture

As we have chosen to work with Dialogflow and its webhook integration, we will have to build a server that responds to POST requests and then forwarding the results again to Dialogflow, who will act as a middle man in the communication between the client and our server. It could also work without a server[6], but using a server has two main advantages:

- **Privacy:** As the bot will have access to sensitive data, we access from our secure locations.
- **Customization:** We will be able to customize the behavior. When working serverless there are limitations by the system on what you can do.

3.1.2 Interactions

The flow of every interaction will be the following:

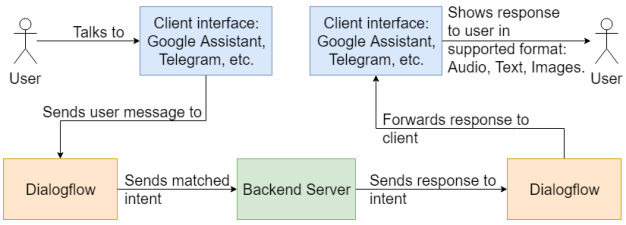

Fig. 1: Complete flow since user inputs until output is received.

In order to make our chatbot more appealing to the users, we are trying to design its responses to look as human as possible as this makes the user to take it more seriously [7]. This will be accomplished by responding using longer phrases and not always sending the same response to the same inputs.

The first challenge designing a chatbot comes up when trying to ask questions to a user. There is one main problem: the agent can only respond to user interactions. To overcome this, we use an ”entry point” which then leads to the question. Then, when the user answers the question and by keeping context of which question is being answered, an intent is fired up. The response from this intent is the next step on the dialog. This is better illustrated on figure 2.

In a chatbot, an intent represents each step of the conversation, from the user input to the bot response. It is the basic unit and it is what makes interactions possible. As intents start with the user input, by themselves they don’t know anything else about the conversation. We have to give context to them. Setting contexts to the intents we can then build very complex interactions, only firing up intents that are enabled in a specific context.

Now that we have defined the base for every interaction, we have to define the interactions that the chatbot will be able to respond to:

- **Welcome.** When user comes back the bot should answer with some greeting.
- **Sign in.** The user has to be enabled by the doctor to use the chatbot.

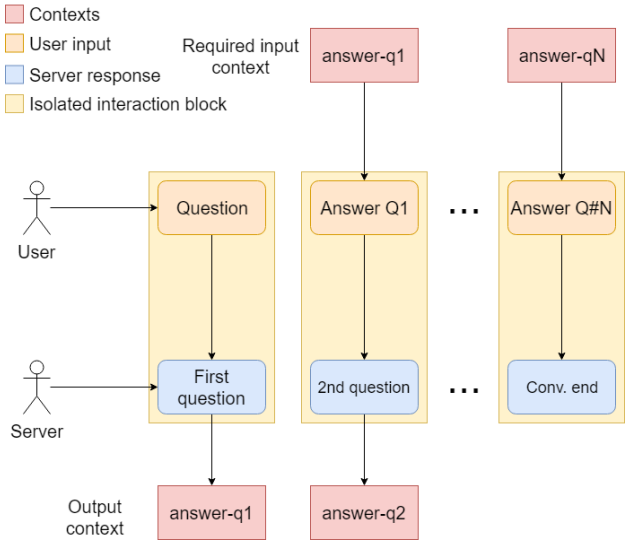

Fig. 2: Message passing from one intent to another and how context is kept.

- **Set current smoking status.** The bot will have to know if the user has already quit smoking or not yet. This can be changed later.
- **Set, update and view stats.** To do simple statistics such as savings.
- **Natural conversation.** The bot will have a more open scenario, where the user can tell it how is feeling.
- **Perform tests.** The patient will be able to do some tests to know more about himself.

We will need a sign in step as we are building a chatbot that integrates with an already existing platform in which the doctors have records of the patients. The sign in will connect to this platform and ensure that the user has been enabled and has permisions to use the chatbot. Also, the tests that we are going to implement are two. This tests are the Fägerstrom test for knowing the patient addiction levels to nicotine[8] and the Richmond test, to have some knowledge about the patient motivation to quit smoking[9].

3.2 Implementation

Here we will discuss how the design proposed in section 3.1.1 has been implemented.

3.2.1 Agent

As we have chosen Dialogflow as the platform to work with, we have to know what do we need in order to make our agent interact with our users. First, we have to implement our conversations as a chain of *Intents*. An intent represents one dialog turn within the conversation and has a number of field to customize its behavior, as can be seen in figure 3

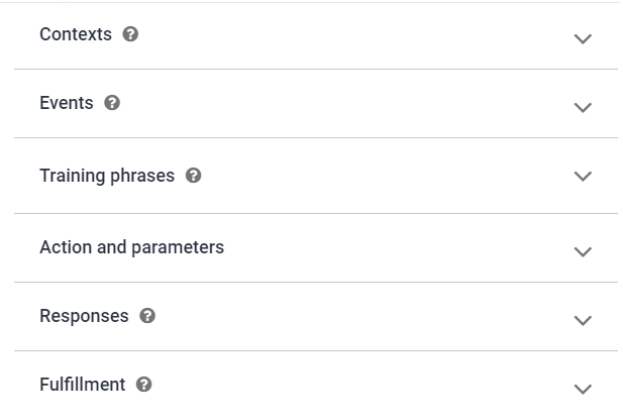

Fig. 3: The structure of an intent and available customizations.

Every intent is isolated from each other. The linking between them is accomplished through *Contexts*. Contexts represent the current state of a user's request and allow the agent to carry information from one intent to another. Using combinations of input and output contexts allows to control the conversational path the user takes through the dialog. Input contexts on an Intent define which contexts have to be active to match the Intent. Output contexts define the contexts that will be active after the intent is processed (on the next user interaction). This allows us to have intents that are fired up by the same phrases but need an specific context, like Yes/No questions. This is widely used in our agent for authenticating the user and creating *restricted* zones.

Then, after contexts are defined we have to define the phrases that will trigger the intent, and what data we need to extract from them. This is what trains the model, so the more example phrases that we can provide, the better the matching will be.

Table 1 shows how input contexts work when matching.

As shown in table 1, we will take advantage of contexts to generate secured intents. At first, the session will have no contexts, and after the user has signed in, the **user-verified** context will be added. Every intent that requires that the user is authenticated, will have **user-verified** as an input and output context.

Session Contexts	Int. Input Contexts	Can match?
No Contexts	No Input Contexts	Yes
No Contexts	likes_cats	No
likes_cats	likes_cats	Yes
likes_cats	No Input Contexts	Yes
likes_cats	likes_dogs	No
likes_cats, likes_dogs	likes_dogs	Yes
likes_cats, likes_dogs	likes_cats, likes_dogs	Yes

TABLE 1: Context match matrix.

Next step is creating the **Training Phrases**. This is what the user has to say in order to fire up the interaction. Dialogflow will try its best to match small variations of the phrases, but the more phrases we define the better will be the result.

We can define *entities*[10] on the training phrases. Entities are parts of the string that Dialogflow will use to extract information and send it as parameters to our webhook. This is very important to create dynamic intents. This way, we define some example training phrases, telling Dialogflow which entities has to look for and then when the user says the same phrase but changing the value of the entity it gets captured. This is demonstrated in figures 4 and 5. This will allow us to ask the user for dates or quantities, in order to do some simple calculations and show statistics or plots.

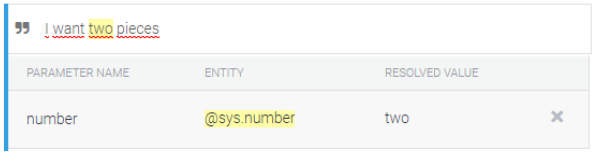

Fig. 4: Training phrase with dynamic parts.

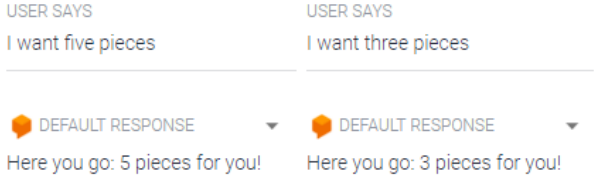

Fig. 5: Value extracted from entity and response customized using it.

After we have defined the contexts and written our training phrases, we have still one thing missing: The responses. There are two options when it comes on how to build the responses.

- **Simple responses:** These are configured from Dialogflow's web interface. The responses here can not perform any kind of logic, just printing the strings that we write. We can use placeholders for the entities or build a Rich Message[11], but that's it. An example is shown in figure 6.

- **Fulfillment:** Enabling fulfillment[12] is as easy as toggling a switch. This will make Dialogflow call an one endpoint after matching and extracting the parameters from the query. Then, the server receives the petition and can process it in any way that is needed: updating a database, sending notifications or emails, etc. Then it talks back to Dialogflow with the customized response.

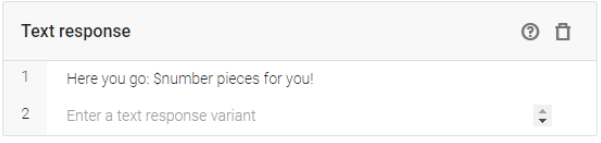

Fig. 6: Defining dynamic responses from the UI.

We will be using fulfillment to process our intents, so we can introduce complex business logic and custom responses to our agent. In figure 7 we can see how adding logic from fulfillment can provide meaningful responses, in comparison to the simpler responses shown on figure 5.

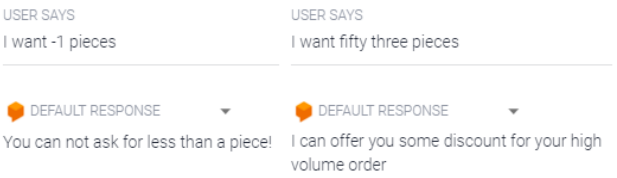

Fig. 7: Adding value to our intent by processing the request with fulfillment.

Finally, after we have completed all the requirements our intent will be ready to work with the chatbot. This work has to be repeated for every intent that we create, which can be a very time-consuming task. A single conversation can have several intents. We can see what a complete Intent would look in figure 8. Because it has input contexts, this intent needs the user to be verified and to be a no smoker at the moment.

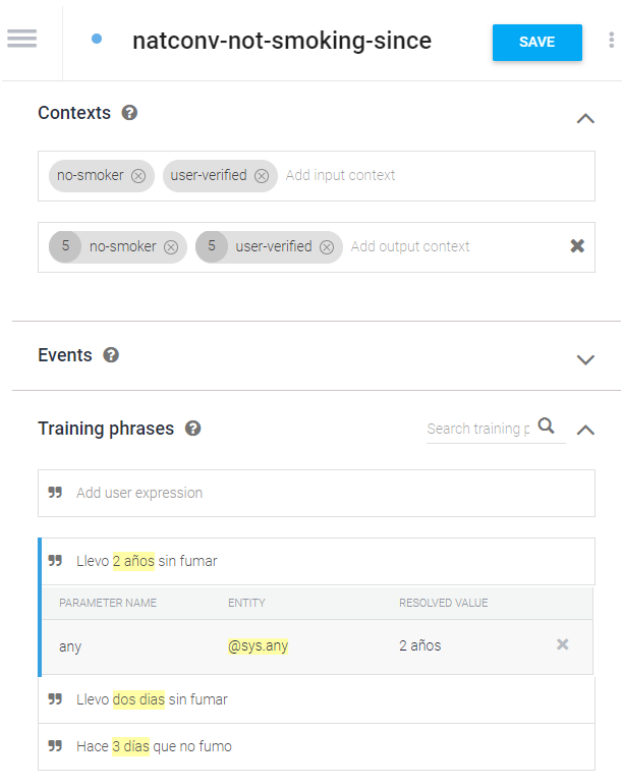

Fig. 8: An intent with input and output contexts and several training phrases with data being extracted from them.

3.2.2 Backend Server

As we want to produce better responses using the fulfillment capabilities shown before, we will need to set up a backend server that is able to receive and process the requests. Dialogflow offers a ready to work fulfillment library for Node.js, which helps building rich responses and sending them back to the agent. If we wanted to implement the backend in another technology we could also integrate directly with the dialogflow API, but then we would have to code again what is already available in the library.

We have chosen to work with the library and stick to Node.js. Node.js is a multiplatform backend server that runs JavaScript code. The code will be written in TypeScript which will allow us to create clean code as we focus in building a multiplatform bot. For now now our chatbot will not have persistent storage of the data, this meaning that each time its restarted, will lose the data. In production, it will be integrated with a database in order to store the data. As Node.js is multiplatform, it does not matter in which OS we run the platform. It has been tested to work in Windows 10 and Ubuntu Server 16.04 LTS.

3.2.2.1 *How intents are handled*: When an intent is matched dialogflow sends a POST request to an arbitrary URL that we have to set. The URL invoked is always the same, and the requests contain the info needed to choose how to process them. The library encapsulates the request in an object so it's easier to access its properties. This object will be the one responsible of handling the intent and sending the response back.

```
function handlePost(request, response) {
    var agent = new WebhookClient({request,
        response});
}
```

Listing 1: Creating the agent handler from the request

After creating the agent object, we have to create the actual methods that will handle the intent. Then we have to map each intent to its handling method and pass the map to the agent.

```
function handlePost(request, response) {
    var agent = new WebhookClient({request,
        response});

    var intentMap = new Map();
    intentMap.set('intent-name', function(
        agent) {
        // Intent handling happens here
    }
    agent.handleRequest(intentMap);
}
```

Listing 2: Creating the agent handler from the request

3.2.2.2 *How multiplatform is achieved*: Now we can handle intents, but we don't know which platform the user activated the agent from. We need a way to implement different messages for different platforms, taking into account each platform limitations: Does it have a screen or is it an audio-only device?

Here is where TypeScript comes to the rescue. The request contains from which platform it has come, so we can create a class for each of our platforms that extend from a super class that contains the intentMap seen in listing 2. This way, following that pattern, we create an instance for each platform and put them in a map. Then when a request arrives, we get the handler instance from the request source, and then get the intentMap from that class.

This allows us to easily add new platforms and have custom implementations for each one if needed. We can also have a base implementation and each subclass can override specific methods. This is very useful to only send visual feedback to those platforms that accept it, but only audio to those which do not, such as Google Home speakers.

4 RESULTS

In this section we are going to show the results that we have obtained after building the bot and show what it can be done. As the chatbot is implemented in Spanish, the content will be shown in Spanish.

The first thing that we wanted was multiplatform. In figure 9 we can see the bot running in two different platforms and the differences that there are between them. On the left, the bot is running under Google Assistant. In this platform we can send the messages to the chatbot as text or as audio. The response will be always in text and if the input was voice, it will be read out loud. On the right, the bot is being used through telegram. This platform input is limited to text only.

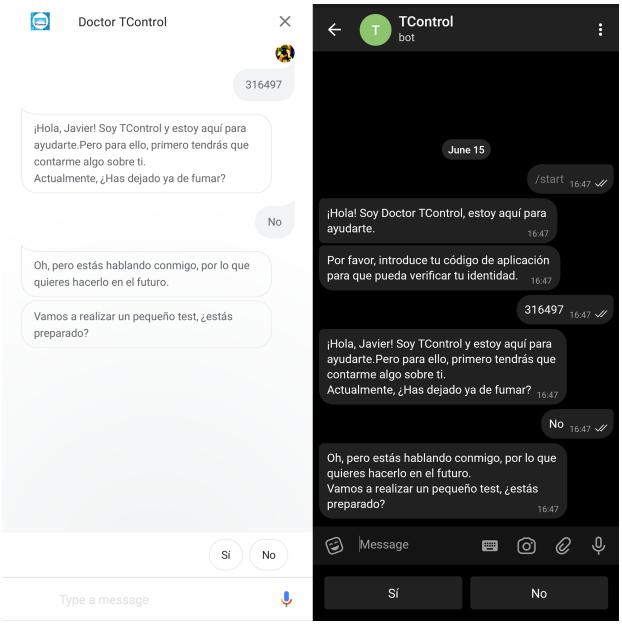

Fig. 9: Starting the bot on Assistant and Telegram.

Another thing we have seen in figure 9 is the user signing in. If an unauthorized user tries to talk to the bot, it does not let him use it unless the user provides a valid code. We can see this behavior on figure 10.

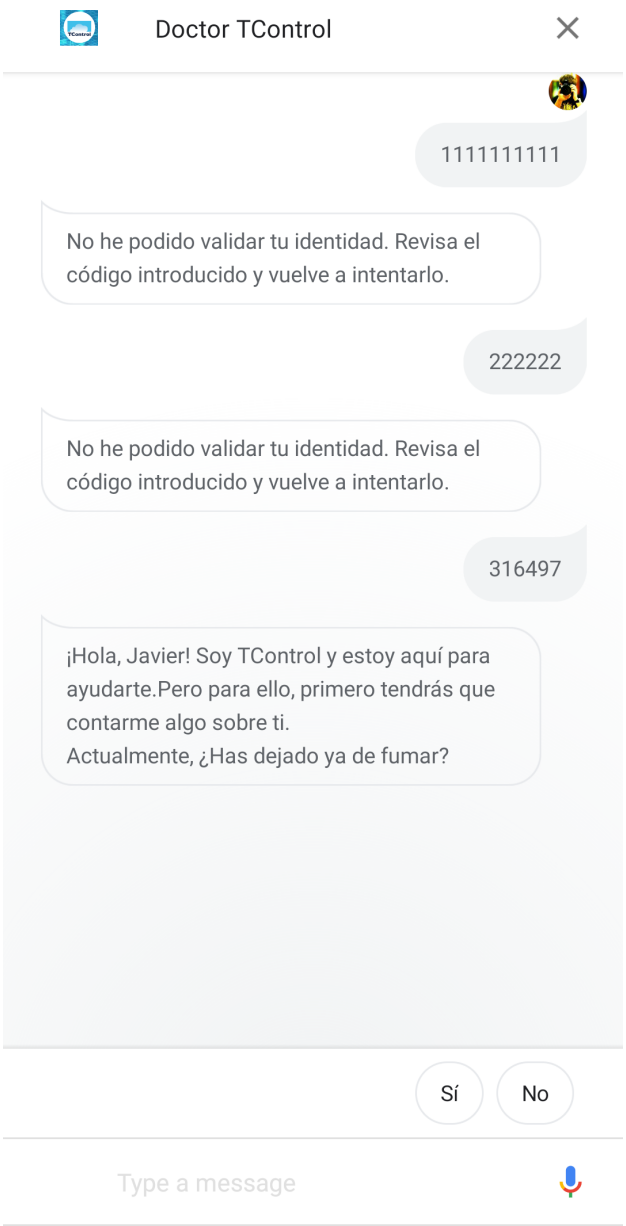

Fig. 10: Only authorized users can access the chatbot.

Following the same principles, with the chatbot now we can track our status: Smoker or Not smoker. This way the chatbot can help us in a different way. If we are in the No-Smoker group, we can:

- **Check out our savings:** When we tell the bot that we are not smoking anymore, it asks for some data that allows it to calculate the savings. If we are in a platform with screen available, it will also print a plot comparing your savings vs the average savings. This can be seen in figure 11.
- **Tell the bot 'I want to smoke':** If you are feeling like smoking, the bot will try and

help you not to smoke.

- **Ask the bot when I stopped smoking:** The bot will answer with the time that has passed since the last smoke.
- **Change our status to Smoker:** If we have had a relapse, we can notify the chatbot and change our status.

Fig. 11: The chatbot can generate plots in real time, providing useful data to the user.

On the other hand, when we are in the Smoker group, we do not have access to the No-Smoker functionality. The No-Smoker group only has a single exclusive intent: Changing the status to no-smoker.

4.1 Discussion

The chatbot has worked correctly in every field. It has all the functionality that was specified as a requirement. Where it has worked better is in smartphones. Visual feedback has proven very useful in order to guide the users through the interactions. Simple yes/no suggestions like the shown in the bottom of figure 10 help the user a lot to understand which kind of answer can be given to a question. This feedback is not available on smart speakers such as the Google Home.

It has shown a great potential as it helps the patient to commit to the task. It also has great NLP and with a good training it does not need the user to say exactly as it was trained, it can understand simple variations of the same phrase and match it to the correct intent.

It has some limitations though. If the user gives an answer that is completely unexpected (the bot has no training or it is very different from the training) it will not match any intent. This makes difficult to create very complex conversations as the tree would grow to have hundreds of leaves very quickly.

The chatbot has great potential if the patients understand what it can do and what it can not do. It is not trying to replace their therapist but being an extra support for the moments in which they need instant help. For more serious matters, the patients should address a real doctor.

5 Conclusions and Future Work

5.1 Conclusions

We have built a chatbot that is able to help a patient quitting smoking in a very little amount of time compared to what it used to cost. Even tough Natural Language Processing has advanced a lot in the last few years, it still has a long way to go. Building simple agents that ask questions with pre-specified answers is really easy and works very well but when this number scales up or the answer is open, the complexity of the agent increases exponentially. To sum up, building a chatbot now is as easy as how well the interactions with the user are planned beforehand.

5.2 Future work

At the moment, the chatbot has a rich feature set, but some improvements that can be done without adding new features to it are:

- **Translating the bot.** It is now working only with spanish inputs and it would increase the potential user base.
- **Extracting insights.** As every interaction is associated to a user, and we control the back-end code, data analysis can be easily added to it.

- **Porting to other platforms.** If Telegram and Google Assistant are not enough, the backend is designed to easily plug in new integrations, so it can work with new platforms that come in the future.

And of course, the creation of new intents or modifying current ones is always a way to improve the chatbot. A further study that continues this work, should test the chatbot against real patients.

REFERENCES

[1] Braun D, Hernandez Mendez A, Matthes F, Langen M. Evaluating Natural Language Understanding Services for Conversational Question Answering Systems. 2017 08;.

[2] Xu A, Liu Z, Guo Y, Sinha V, Akkiraju R. A New Chatbot for Customer Service on Social Media. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. CHI '17. New York, NY, USA: ACM; 2017. p. 3506–3510. Available from: http://doi.acm.org/10.1145/3025453.3025496.

[3] Cameron G, Cameron D, Megaw G, Bond R, Mulvenna M, O ' Neill S, et al. In: Assessing the Usability of a Chatbot for Mental Health Care; 2019. p. 121–132.

[4] e Silva JFO. Interactive bot to support the use of the UPTEC intranet. 2018;.

[5] Hill J, Ford WR, Farreras IG. Real conversations with artificial intelligence: A comparison between human–human online conversations and human–chatbot conversations. Computers in Human Behavior. 2015;49:245 – 250. Available from: http://www.sciencedirect.com/science/article/pii/S0747563215001247.

[6] Yan M, Castro P, Cheng P, Ishakian V. Building a Chatbot with Serverless Computing. In: Proceedings of the 1st International Workshop on Mashups of Things and APIs. MOTA '16. New York, NY, USA: ACM; 2016. p. 5:1–5:4. Available from: http://doi.acm.org/10.1145/3007203.3007217.

[7] Araujo T. Living up to the chatbot hype: The influence of anthropomorphic design cues and communicative agency framing on conversational agent and company perceptions. Computers in Human Behavior. 2018;85:183 – 189. Available from: http://www.sciencedirect.com/science/article/pii/S0747563218301560.

[8] HEATHERTON TF, KOZLOWSKI LT, FRECKER RC, FAGERSTROM KO. The Fagerström Test for Nicotine Dependence: a revision of the Fagerstrom Tolerance Questionnaire. British Journal of Addiction. 1991;86(9):1119–1127. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1360-0443.1991.tb01879.x.

[9] RICHMOND RL, KEHOE LA, WEBSTER IW. Multivariate models for predicting abstention following intervention to stop smoking by general practitioners. Addiction. 1993;88(8):1127–1135. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1360-0443.1993.tb02132.x.

[10] DialogFlow. *Entities overview*;. [Online]. Available from: https://dialogflow.com/docs/entities.

[11] DialogFlow. *Rich messages*;. [Online]. Available from: https://dialogflow.com/docs/intents/rich-messages.

[12] DialogFlow. *How fulfillment works*;. [Online]. Available from: https://dialogflow.com/docs/fulfillment/how-it-works.